## Dynamic IP Address Filtering for Enhanced Healthcare Data Security

## Project description

## To address the request the requests from the security team of an healthcare company, it’s the responsibility of the team to ensure that only authorized employees can access restricted content, especially when dealing with personal patient records. This project involved creating a Python algorithm to automatically update the allow list of IP addresses that are permitted to access a secure subnetwork. The goal was to identify and remove any IP addresses listed on a remove list from this allow list, ensuring that only current authorized personnel can access sensitive information. By doing so, I streamlined the process of managing network access, enhancing our organization’s security posture.

## Open the file that contains the allow list

In this step, I needed to open the file named `allow\_list.txt`, which contains the list of IP addresses allowed to access the restricted subnetwork. To accomplish this, I used a `with` statement to safely open the file and assigned it to a variable called `file`. This method ensures that the file is properly closed after I finish working with it, even if an error occurs. Here’s the code I used:

# Assign `import\_file` to the name of the file

import\_file = "allow\_list.txt"

# Build `with` statement to open the file

with open(import\_file, "r") as file:

    # Additional steps will be performed here

## Read the file contents

Once I opened the file, the next step was to read its contents. To do this, I used the `.read()` method, which converts the file’s contents into a string. I stored this string in a variable called `ip\_addresses`, making it easier to manipulate the data in subsequent steps. Here’s the steps that I followed:

# Use `.read()` to read the imported file and store it in a variable named `ip\_addresses`

ip\_addresses = file.read()

# Display `ip\_addresses`

print(ip\_addresses)

## Convert the string into a list

After reading the file contents into a string, I needed to convert this string into a list of individual IP addresses. This was essential because it allowed me to manipulate each IP address separately. I used the `.split()` method, which splits the string into a list based on spaces or new lines. Here’s the code I wrote:

# Use `.split()` to convert `ip\_addresses` from a string to a list

ip\_addresses = ip\_addresses.split()

# Display `ip\_addresses`

print(ip\_addresses)

## Iterate through the remove list

With the allow list now in a list format, I needed to check each IP address against the remove list. I set up a `for` loop to iterate through the list of IP addresses, using `element` as the loop variable. This allowed me to evaluate each IP address individually to determine if it should be removed. Here’s the iteration setup:

# Build iterative statement

# Name loop variable `element`

# Loop through `ip\_addresses`

for element in ip\_addresses:

    # Display `element` in every iteration

    print(element)

## Remove IP addresses that are on the remove list

During each iteration, I checked if the current IP address (`element`) was in the `remove\_list`. If it was, I used the `.remove()` method to delete it from the `ip\_addresses` list. The `.remove()` method is effective here because I ensured there are no duplicate IP addresses in the list, so I don’t need to worry about accidentally removing more than necessary. Here’s the code I implemented:

# Build conditional statement

# If current element is in `remove\_list`,

if element in remove\_list:

    # then current element should be removed from `ip\_addresses`

    ip\_addresses.remove(element)

# Display `ip\_addresses`

print(ip\_addresses)

## Update the file with the revised list of IP addresses

After removing the necessary IP addresses, I converted the updated list back into a string using the `.join()` method, placing each IP address on a new line. Finally, I used another `with` statement and the `.write()` method to overwrite the original `allow\_list.txt` file with the revised list of IP addresses. This step ensured that the file now reflects the current authorized IP addresses. Here’s the code for this final step:

# Convert `ip\_addresses` back to a string so that it can be written into the text file

ip\_addresses = "\n".join(ip\_addresses)

print(ip\_addresses)

# Build `with` statement to rewrite the original file

with open(import\_file, "w") as file:

    # Rewrite the file, replacing its contents with `ip\_addresses`

    file.write(ip\_addresses)

## Full Code Implementation and Results

The following section presents the complete Python code that implements the algorithm I designed for updating the allow list of IP addresses. This algorithm was created to ensure that IP addresses listed in the `remove\_list` are effectively removed from the `allow\_list.txt` file. The process involves reading the contents of the file, converting the IP addresses into a list, filtering out the unwanted addresses, and finally writing the updated list back to the file.

Below is the full implementation:

# The remove\_list contains IP addresses to be removed from the allow list.

remove\_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]

# Open the file and read the contents

with open("allow\_list.txt", "r") as file:

    ip\_addresses = file.read()

# Convert the string of IP addresses into a list

ip\_addresses = ip\_addresses.split()

# Remove the IP addresses that are in the remove list

for element in ip\_addresses:

    if element in remove\_list:

        ip\_addresses.remove(element)

# Convert the list back to a string and write it to the file

ip\_addresses = "\n".join(ip\_addresses)

# Updating the file with the revised list

with open("allow\_list.txt", "w") as file:

    file.write(ip\_addresses)

# Printing the updated content of the allow list

print(ip\_addresses)

Results

Upon executing the code above with the ‘allow\_list.txt’ file included in my repository, the `allow\_list.txt` file is updated to exclude any IP addresses that were listed in the `remove\_list`. The revised content of the file is as follows:

ip\_address

192.168.25.60

192.168.205.12

192.168.6.9

192.168.52.90

192.168.90.124

192.168.186.176

192.168.133.188

192.168.203.198

192.168.218.219

192.168.52.37

192.168.156.224

192.168.60.153

192.168.69.116

This result confirms that the IP addresses identified for removal have been successfully deleted from the allow list, ensuring that only the approved addresses remain.

## Summary

In this project, I developed a Python algorithm to efficiently manage and update a list of authorized IP addresses for a restricted subnetwork within a healthcare organization. The algorithm opened the existing allow list, read its contents, and converted them into a list format.

By iterating through the list of IP addresses and comparing each one with a remove list, I was able to automatically remove any unauthorized addresses from the list.

The final step involved updating the original file with the revised list, ensuring that only authorized personnel have access to sensitive patient data. This process not only enhances the security of the organization but also automates a critical part of the access management workflow.